Journal of Power Sources 216 (2012) 530–541

Contents lists available at SciVerse ScienceDirect

Journal of Power Sources

journal homepage: www.elsevier.com/locate/jpowsour

A new stochastic algorithm for proton exchange membrane fuel cell stack design optimization

Uttara Chakraborty [a,b,*]

[a] 172 San Angelo Drive, Chesterfield, Missouri 63017, USA
[b] Department of Science and Mathematics, Maryville University, 650 Maryville University Drive, St. Louis, Missouri 63141, USA

HIGHLIGHTS

► Develops a new, original algorithm for PEM fuel cell stack design optimization.
► Solves a PEMFC problem that is analytically intractable and computationally hard.
► Outperforms three state-of-the-art methods on three performance metrics.
► Claims of improved results are statistically validated.
► A Markov chain analysis of the algorithm is provided.

ARTICLE INFO

Article history:
Received 29 February 2012
Received in revised form
8 June 2012
Accepted 9 June 2012
Available online 19 June 2012

Keywords:
Proton exchange membrane fuel cell
PEMFC stack
Optimization
Heuristic
Simulation
Markov chain

ABSTRACT

This paper develops a new stochastic heuristic for proton exchange membrane fuel cell stack design optimization. The problem involves finding the optimal size and configuration of stand-alone, fuel-cell-based power supply systems: the stack is to be configured so that it delivers the maximum power output at the load's operating voltage. The problem apparently looks straightforward but is analytically intractable and computationally hard. No exact solution can be found, nor is it easy to find the exact number of local optima; we, therefore, are forced to settle with approximate or near-optimal solutions. This real-world problem, first reported in *Journal of Power Sources 131*, poses both engineering challenges and computational challenges and is representative of many of today's open problems in fuel cell design involving a mix of discrete and continuous parameters. The new algorithm is compared against genetic algorithm, simulated annealing, and (1+1)-EA. Statistical tests of significance show that the results produced by our method are better than the best-known solutions for this problem published in the literature. A finite Markov chain analysis of the new algorithm establishes an upper bound on the expected time to find the optimum solution.

1. Introduction

Fuel cells (FCs) are an attractive energy source with a great promise for efficiently and economically meeting the energy needs in a wide variety of applications. Proton exchange membrane fuel cells (also called polymer electrolyte membrane fuel cells) or PEMFCs are a fast-growing alternative for distributed generation sources (e.g., [5]). PEM fuel cells, unlike other distributed generation technologies such as wind or photovoltaic systems, have the advantage of being suitable for placement at any site in a distribution system. PEM fuel cells also find major applications in electric vehicles, as these cells produce very few pollutants.

Fuel cells are expensive devices, and their design is an engineering challenge [5,11]. In most real-world applications, to meet specific output requirements, a number of fuel cells are arranged in a particular (electrical) configuration. A series connection of cells, leading to a stack, is fairly common. It is possible, depending on requirements and specifications, to connect cells in series and/or parallel. The current from a cell can be increased by increasing the cell surface area.

The following problem is addressed in this paper: Find optimal values for (i) the number of cells to be placed in series in a stack, (ii) the number of such stacks to be connected in parallel, and (iii) the cell area, such that the PEMFC configuration delivers the rated voltage at the rated power while the cost of building it is minimum [14]. The problem apparently looks straightforward but is analytically intractable and computationally hard. No exact solution can be found, nor is it easy to find the exact number of local optima; we, therefore, are forced to settle with approximate or near-optimal

* 172 San Angelo Drive, Chesterfield, Missouri 63017, USA. Tel.: +1 314 3179805.
E-mail address: uchakra@yahoo.com.

0378-7753/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.jpowsour.2012.06.040

Nomenclature

N_s	number of series cells in a group
N_p	number of groups in parallel
E_{Nernst}	Nernst potential of a single cell, V
$E_{0,cell}$	ideal standard potential of a single cell, V
A	Tafel slope, V
B	concentration loss constant, V
V_{cell}	output terminal voltage of a single PEMFC, V
V_{stack}	output terminal voltage of the PEMFC stack, V
$V_{load,rated}$	rated output terminal voltage of the PEMFC stack, V
$P_{load,rated}$	rated output power of the PEMFC stack, W
$P_{load,max}$	maximum output power of the PEMFC stack, W
$V_{load,maxpp}$	output voltage at maximum power point of the PEMFC stack, V
A_{cell}	cell area, cm^2
i_{den}	current density, A cm^{-2}
$i_{0,den}$	exchange current density, A cm^{-2}
$i_{load,den}$	load current density, A cm^{-2}
$i_{limit,den}$	limiting current density, A cm^{-2}
$i_{n,den}$	crossover and internal current density, A cm^{-2}
r_{area}	area-specific resistance of a single cell, KΩ cm^2
p_{H_2}	partial pressure of hydrogen, atm
p_{O_2}	partial pressure of oxygen, atm
α	charge transfer coefficient; also, constant in the new algorithm
K_{num}, K_{vdiff}, K_{area}	coefficients in cost function
η_{act}	activation loss, V
η_{conc}	concentration loss, V
η_{ohm}	ohmic loss, V

solutions. This real-world problem poses both engineering challenges and computational challenges and is representative of many of today's open problems in fuel cell design that involve a mix of discrete and continuous parameters. This problem is computationally hard, in that it is difficult to find a mathematically well-behaved formulation amenable to standard (e.g., derivative-based) optimization techniques. Heuristics, therefore, are a viable alternative in this case.

The use of optimization algorithms for producing fuel cell stack designs is an emerging field of research. The best-known method for solving this problem was published in Ref. [14] where a genetic algorithm was used to find optimal solutions. The present paper develops a new heuristic algorithm that is competitive with genetic algorithm, simulated annealing, (1+1)-EA, and the best result of Ref. [14].

The remainder of this paper is organized as follows. Section 2 describes the PEMFC electrochemical model and presents the main equations needed for solving the design problem. Section 3 defines the cost function and describes the new algorithm, including how the novel transition probability function helps the search process. Section 4 presents a Markov chain-based analysis of the algorithm. Section 5 presents simulation results along with statistical tests of significance. Conclusions are drawn in Section 6.

2. The design problem

The overall reaction in a PEM fuel cell is given by

$$H_2 + \frac{1}{2}O_2 = H_2O.$$

The thermodynamic or reversible potential (emf), E_{Nernst}, of a single PEM fuel cell is given by the Nernst equation [11]:

$$E_{Nernst} = E_{0,cell} + \frac{RT}{2F}\ln\left(p_{H_2}\sqrt{p_{O_2}}\right) \quad (1)$$

where $E_{0,cell}$ is the standard reference potential of a single cell.

The following types of losses (or voltage drops, or irreversibilities) are generally considered in the literature (see, e.g., [11,3]):

- Activation loss caused by the slowness of the electrochemical reactions taking place on the surface of the electrode [6]:

$$\eta_{act} = \frac{RT}{\alpha nF}\ln\left(\frac{i_{den}}{i_{0,den}}\right)$$

where α is the electron transfer coefficient, n is the number of electrons (per mole) participating in the reaction, i_{den} is the current density, and $i_{0,den}$ the exchange current density (with $i_{den}>i_{0,den}$). Introducing the Tafel constant (Tafel slope)

$$A = \frac{RT}{\alpha nF},$$

this loss can be expressed as

$$\eta_{act} = A\ln\left(\frac{i_{den}}{i_{0,den}}\right). \quad (2)$$

- Concentration loss (or mass transport loss) that results from the change in concentration of the reactants at the surface of the electrodes as the fuel is used:

$$\eta_{conc} = -\frac{RT}{nF}\ln\left(1 - \frac{i_{den}}{i_{limit,den}}\right)$$

where n is the number of electrons transferred per mole, i_{den} is the current density, and $i_{limit,den}$ the limiting current density. This can also be expressed as

$$\eta_{conc} = -B\ln\left(1 - \frac{i_{den}}{i_{limit,den}}\right) \quad (3)$$

where

$$B = \frac{RT}{nF}$$

is the concentration loss constant.

- Ohmic loss due to the electrical resistance of the electrodes, the polymer membrane, and the conducting resistance between the membrane and the electrodes. This (area-specific) resistance, represented by r_{area}, depends upon current and temperature. The ohmic loss, η_{ohm}, is given by

$$\eta_{ohm} = i_{den}r_{area}. \quad (4)$$

- Loss due to fuel crossover (waste of fuel passing through the electrolyte) and internal current. This loss can be accounted for by introducing an additional component of current, called the fuel crossover and internal current.

Combining all the four losses, the terminal voltage, V_{cell}, of a single cell is given by

$$V_{\text{cell}} = E_{\text{Nernst}} - A\ln\left(\frac{i_{\text{den}} + i_{\text{n,den}}}{i_{0,\text{den}}}\right) + B\ln\left(1 - \frac{i_{\text{den}} + i_{\text{n,den}}}{i_{\text{limit,den}}}\right) - \left(i_{\text{den}} + i_{\text{n,den}}\right) r_{\text{area}}$$

where the expressions for η_{act} (equation (2)), η_{conc} (equation (3)), and η_{ohm} (equation (4)) have been modified to include $i_{\text{n,den}}$, the fuel crossover and internal current density. Now, for N_s cells connected in series, the terminal (load) voltage of the combination is given by

$$V_{\text{series}} = N_s V_{\text{cell}},$$

and the corresponding load current density is i_{den}. When N_p such groups are connected in parallel, the terminal (load) voltage remains the same, namely V_{series}, but the load current is multiplied N_p times.

If $i_{\text{load,den}}$ represents the load current density for the entire stack, the stack terminal voltage is given by

$$\begin{aligned} V_{\text{stack}} = N_s\Bigg\{ & E_{\text{Nernst}} - A\ln\left(\frac{i_{\text{load,den}}/N_p + i_{\text{n,den}}}{i_{0,\text{den}}}\right) \\ & + B\ln\left(1 - \frac{i_{\text{load,den}}/N_p + i_{\text{n,den}}}{i_{\text{limit,den}}}\right) \\ & - \left(i_{\text{load,den}}/N_p + i_{\text{n,den}}\right) r_{\text{area}} \Bigg\}. \end{aligned} \tag{5}$$

For a particular problem, the rated load voltage and the rated power are specified in the problem statement. The task is to minimize the number of cells in series, the number of cells in parallel, and the cell area such that the rated load voltage at maximum power point is 12 V and maximum power is at least 200 W (corresponding to 730 kWh per year). These requirements come from a research project on designing a power supply system to provide dc electricity to a single dwelling in a remote area where a physical PEMFC stack is to be installed in a family house [14].

3. Methodology

3.1. The new algorithm

Our heuristic is a point-based (not population-based) search that starts out with a single trial solution (we use the terms *point*, *trial solution*, *vector*, and *state* interchangeably) in the state space and samples other points in a stochastically guided manner, to arrive at an optimal or near-optimal solution. At each iteration of the algorithm, a mutant of the current point is created by random perturbation, and a decision is made on whether or not to accept the mutant as the next point. The algorithm stops when either a pre-determined number of iterations are completed or a solution of acceptable quality is found, whichever happens earlier. Our algorithm employs a form of controlled greediness: a transition to a better point is accepted deterministically, while a transition to a worse point is implemented conditionally, with a probability that is calculated as a function of the costs (objective functions) of the current point and the mutant. For a worse move, the cost difference d, given by the mutant's cost minus the current cost, is always positive for a minimization problem. We argue that the algorithm should explore the search space vigorously at the beginning of the process, mellowing down to a fine-tuning phase as the process matures. This leads us to formulate the two core requirements that the algorithm must satisfy:

- At a given iteration t, transition probabilities $prob_1$ and $prob_2$, corresponding, respectively, to cost differences d_1 and d_2, where $0 < d_1 < d_2$, should obey the relationship $prob_1 > prob_2$.
- The same cost difference $d > 0$ should produce a higher transition probability at iteration t_1 than at iteration t_2, where $t_1 < t_2$.

These two requirements lead us to design a new transition probability function. An estimate of the expected cost difference between a given point and its mutant (a mutant is obtained by random perturbation of the original point; see Section 3.4) is produced from the average cost-difference of a pre-determined number of pairs of randomly generated points and their respective mutants. This estimate is used in the transition probability function. Of all the points evaluated in the expected-difference estimation process, the best point is used as the start point of the search. In order to emphasize an intense exploration of the search space during the early iterations, the algorithm uses a very high transition probability (close to 1) for a specified number of iterations at the beginning of a run. For the rest of the run, the following function is used:

$$prob = \frac{\alpha^{evals}}{1 + (d/davg)^2}$$

where *evals* is the number of cost function evaluations consumed so far, α is a constant (slightly less than one) and *davg* represents the estimated difference. The complete pseudo-code is given in Section 3.7.

3.2. Solution representation

A PEMFC design is encoded as a three-component vector representing the number of cells in series N_s, the number of parallel groups N_p, and the cell area A_{cell}.

3.3. Initial point

As mentioned in Section 3.1, the search is started with an initial point that is selected as the best of all the points used in the expected cost difference estimation stage. Each point created in the expected cost difference estimation stage has its three components set to values chosen uniformly randomly from pre-determined lower and upper bounds (Table 1) of each parameter.

3.4. Mutant generation

For creating a mutant from a given vector, a small, randomly determined, amount is algebraically added to each parameter in the vector. Since the process is probabilistic, the number of parameters mutated in a single vector in a single iteration can vary from zero (no parameter mutated) to three (all parameters mutated). The pseudocode for mutating *oldValue* into *newValue* for each of the three parameters is given below ($0 < x < 1$; $u(0,1)$ returns a uniform random number between zero and one; *hi* and *lo* represent the upper and lower bounds, respectively):

```
1. $v = x*(hi - lo)*u(0,1)$;
2. With 50% probability, set $v = -v$;
3. $newValue = oldValue + v$;
4. if $newValue < lo$
     begin
     if $oldValue = lo$
```

Table 1
Lower and upper bounds of design variables.

Varible to be optimized	Lower bound	Upper bound
Number of cells in series	1	50
Number of parallel stacks	1	50
Cell area (cm^2)	10	400

```
        $newValue = lo$
        else $newValue = lo + u(0,1)*(oldValue - lo)$;
        end
        else if $newValue > hi$
                begin
                if $oldValue = hi$
                $newValue = hi$
                else $newValue = hi - u(0,1)*(hi - oldValue)$;
                end
```

3.5. Objective function

The problem is formulated as one of minimization. The cost (objective) function is deterministic and is given by a linear combination (weighted sum) of four contributing terms. Deviations, if any, from the rated load voltage and the rated power are made to add a positive term to the cost. The maximum power ($P_{\text{load,max}}$) and the load voltage at maximum power point ($V_{\text{load,maxpp}}$) are found out numerically (through iterations over load current) from equation (5) (the PEMFC parameter values are listed in Table 2). The cost of a solution vector (N_s, N_p, A_{cell}) is computed as:

$$K_{\text{num}}*N_p*N_s + K_{\text{vdiff}}*\left|V_{\text{load,rated}} - V_{\text{load,maxpp}}\right| + K_{\text{area}}*A_{\text{cell}} + \text{penalty}$$

where

$$\text{penalty} = \begin{cases} \text{PENALTY}-\text{LOW}-\text{POWER}: & P_{\text{load,max}} < P_{\text{load,rated}} \\ 0: & \text{otherwise} \end{cases}$$

The values of the coefficients and other constants are given in Table 3.

3.6. Adaptive penalty

In the preceding sub-section, the penalty for violation of the rated power was fixed and static: the penalty value was not dependent on the amount by which the power fell short of the rated power, nor did it change with the progression of the search process. An obvious advantage of using a fixed penalty is the reduced computational burden. A variant of the new heuristic can be defined by making the algorithm adapt the penalty as the search progresses. We achieve that by designing a penalty function that draws inspiration from Ref. [2] (and also Ref. [13]) where an adaptive penalty function was proposed for a genetic algorithm. Most of the penalty adaptation methods in the literature (e.g., [7], [1], [15]) deal with population-based searches, where the penalty function involves a metric (such as an average or the best value) that is computed over the entire population. Our algorithm, however, does not use a population, and therefore standard adaptive penalty approaches cannot be used. We multiply by a coefficient the difference between the algorithm-generated power and the rated power and use the product as the penalty value. The coefficient is updated every k iterations (where k is a pre-determined constant) depending on whether or not all of the most-recent k points were feasible (a feasible point is one with a $P_{\text{load,max}}$ of at least 200 Watts). Letting c denote the coefficient and t the iteration number, the update schedule can be described as follows:

```
if all the intervening $k$ points between iterations t and $t+k$ are feasible
then $c(t+k) = c(t)k_1$  //decrease
else if all the intervening $k$ points are infeasible
    then $c(t+k) = c(t)k_2$  //increase
    else $c(t+k) = c(t)$  //no change
```

In between two consecutive update attempts, the c value remains unchanged. The three constants $k \geq 1, 0 < k_1 < 1, k_2 > 1$ are parameters of the adaptive penalty scheme and are to be determined in advance. For this paper, we choose $k = 5$, $k_1 = 0.9$, and $k_2 = 1.1$ (Table 3).

Table 2
PEMFC single cell parameter values.

Parameter	Value
E_{Nernst}	1.04 V
r_{area}	98.0e-06 KΩ cm^2
$i_{\text{n,den}}$	1.26 mA cm^{-2}
$i_{\text{limit,den}}$	129 mA cm^{-2}
A	0.05 V
B	0.08 V
$i_{0,\text{den}}$	0.21 mA cm^{-2}

Table 3
Values of coefficients and constants.

Parameter	Value
$V_{\text{load,rated}}$	12 V
$P_{\text{load,rated}}$	200 Watts
K_{num}	0.5
K_{vdiff}	10
K_{area}	0.001
PENALTY-LOW-POWER	200
k	5
k_1	0.9
k_2	1.1

3.7. Algorithm pseudocode

The following symbols are used in the pseudocode description:

Number of evaluations of the cost function	*evals*
Maximum allowable number of evaluations of the cost function	*maxEvals*
Size of the initial pool of trial solutions for average difference estimation	*poolSize*
Transition probability	*prob*
The pre-determined fraction of *maxEvals*, used in computing *prob*	*evalFrac*
Average (positive) difference in cost function	*davg*
Current solution	*current*
Mutant to current solution	*next*
Best-so-far solution in a single run	*best*
Desired or target cost	*targetCost*

Algorithm for cost minimization:

```
1. Set $evals = 0$;
2. Randomly initialize (Section 3.3) $poolSize/2$ trial solutions; by random perturbation, generate a mutant of each of these trial solutions;
3. Find $davg$ from $poolSize/2$ cost differences;
4. Set $current$ = the best of the trial solutions used in Step 2;
5. Set $best = current$ ;
6. While cost of $best \geq targetCost$ and $evals \leq maxEvals$ do
   (a) Create a mutant, $next$, of $current$ ;
   (b) If cost of $next <$ cost of $best$
       $best = next$;
   (c) If cost of $next <$ cost of $current$
       $current = next$
       else
        i. Set $d$ = cost of $next$ - cost of $current$;
```

```
    ii. If $evals \le evalFrac \times maxEvals$
        $prob = \alpha^{\mathrm{evals}}$
        else $prob = \dfrac{\alpha^{\mathrm{evals}}}{1 + (d/davg)^2}$;
    iii. With probability $prob$, set $current = next$;
7. Output $best$ as the solution;
```

4. Markov chain analysis of the algorithm

Our algorithm implements a Markov chain [9] of vectors (or states) (we use the words vector and state interchangeably), because the probability of transition from a given state onto another state is dependent on only the current state and the evaluation count and is independent of the history of the traversal through the states. The Markov chain is inhomogeneous, as the transition probabilities depend on the number of evaluations.

Let $M(i)$ represent the set of all possible mutants of vector (or state) i. Since the cell area parameter is inherently continuous (real-valued), it is in principle possible to generate an infinite number of mutants from a given vector. In a computer program, however, because of the finite precision of the computer, the total number of possible cell area values is extremely large but finite. Therefore we assume that $|M(i)|$ is finite.

Now, in a single traversal of the loop in Step 6 of our algorithm, at evaluation number n, the probability $p_{\mathrm{ij}}(n)$ that state i transitions in one step to state j is given by the product of two probabilities:

$$p_{\mathrm{ij}}(n) = p_{\mathrm{select}}(j|M(i)) \times p_{\mathrm{tran,ij}}(n)$$

where $p_{\mathrm{select}}(j|M(i))$ represents the probability that state j is selected from among the members of the set $M(i)$, and $p_{\mathrm{tran,ij}}(n)$ is the (one-step) transition probability that the move from i to j takes place at evaluation n. Assuming the states in $M(i)$ to be equally likely to be generated, we have

$$p_{\mathrm{select}}(j|M(i)) = \frac{1}{|M(i)|}.$$

The p_{ij} entries are then given by:

$$p_{ij}(n) = \begin{cases} 0 & \text{if } i \ne j, j \notin M(i) \\ \dfrac{1}{|M(i)|} & \text{if } i \ne j, j \in M(i), cost(j) < cost(i) \\ \dfrac{\alpha^{\mathrm{n}}}{|M(i)|} & \text{if } i \ne j, j \in M(i), cost(j) \ge cost(i), n \le evalFrac \times \mathrm{max}Evals \\ \dfrac{\alpha^{\mathrm{n}}}{|M(i)|\left(1 + \left(\dfrac{d}{davg}\right)^2\right)} & \text{if } i \ne j, j \in M(i), cost(j) \ge cost(i), n > evalFrac \times \mathrm{max}Evals \end{cases}$$

and

$$p_{\mathrm{ii}}(n) = 1 - \sum_{\substack{j \in M(i) \\ i \ne j}} p_{\mathrm{ij}}(n).$$

The state transition policy in Step 6 of our algorithm causes any state to be reachable from any other state, given enough intermediate transitions.

Let $p_{ij}^{(k)}(n)$ denote the probability of transition from state i to state j in k steps, starting at evaluation n for the first transition from state i. Let $f_{ij}^{(k)}(n)$ denote the probability that starting from state i at evaluation n, the chain enters state j *for the first time* at the kth step. If $T_{\mathrm{ij,n}}$ is a random variable representing the first hitting time of state j starting, at evaluation n, from state i, then

$$\mathrm{Probability}(T_{ij,n} = k) = f_{ij}^{(k)}\left(n\right), \tag{6}$$

and

$$p_{ij}^{(k)}(n) = \sum_{m=1}^{k} f_{ij}^{(m)}(n) p_{jj}^{(k-m)}(n+m), \tag{7}$$

where

$$f_{ij}^{(1)}(n) = p_{ij}^{(1)}(n) = p_{ij}(n) \text{ for all } i, j, \tag{8}$$

$$f_{ij}^{(0)}(n) = p_{ij}^{(0)}(n) = 0 \text{ for } i \ne j, \tag{9}$$

$$f_{ii}^{(0)}(n) = p_{ii}^{(0)}(n) = 1. \tag{10}$$

The mean (expected) first hitting time of state j, starting, at evaluation n, from state i is then given by

$$E(T_{ij,n}) = \sum_{k=1}^{\infty} k \cdot f_{ij}^{(k)}(n). \tag{11}$$

An upper bound on $E(T_{ij,n})$ can be found as follows. Following our definitions,

$$f_{ij}^{(k)}(n) = \sum_{s_1, s_2, \ldots, s_{k-1} \ne j} p_{is_1}(n) \times p_{s_1 s_2}(n+1) \times p_{s_2 s_3}(n+2) \times \cdots \times p_{s_{k-1} j}(n+k-1) \tag{12}$$

where the sum is over all length-k paths from i to j. Therefore

$$\begin{aligned}
p_{ij}^{(k)}(n) &\geq p_{is_1}(n)\times p_{s_1s_2}(n+1)\times p_{s_2s_3}(n+2)\times\cdots\times p_{s_{k-1}j}(n+k-1)\\
&\geq p_{is_1}(n+k-1)\times p_{s_1s_2}(n+k-1)\times p_{s_2s_3}(n+k-1)\times\cdots\\
&\quad\times p_{s_{k-1}j}(n+k-1)\\
&=\prod_{y,z} p_{yz}(n+k-1)\\
&\geq\left(\min_{y,z}\left\{p_{yz}(n+k-1)\right\}\right)^k\\
&\geq\left(\min_{y,z}\left\{\frac{\alpha^{n+k-1}}{|M(y)|\left(1+\left(\frac{d_{yz}}{davg}\right)^2\right)}\right\}\right)^k\\
&=\left(\frac{\alpha^{n+k-1}}{\mu\left(1+\left(\frac{\delta}{davg}\right)^2\right)}\right)^k
\end{aligned}$$

where

$$\mu = \max_y\{|M(y)|\} \tag{13}$$

and

$$\delta = \max_{y,z}\{d_{yz}\}. \tag{14}$$

If θ represents the maximum possible number of cost evaluations to be used and if γ is the maximum number of steps needed to reach a state starting from some other state (excluding re-visits of states), then the probability that state j will ever be visited starting from state i is at least $\left(\frac{\alpha^\theta}{\mu\left(1+\left(\frac{\delta}{davg}\right)^2\right)}\right)^\gamma$. Thus the expected hitting time of state j is less than or equal to $1/\left(\frac{\alpha^\theta}{\mu\left(1+\left(\frac{\delta}{davg}\right)^2\right)}\right)^\gamma$ or $\left(\frac{\mu\left(1+\left(\frac{\delta}{davg}\right)^2\right)}{\alpha^\theta}\right)^\gamma$. Without loss of generality, we can assume state j to be the global optimum state; then the above upper bound applies to the expected time to find the global optimum for the first time.

5. Simulation results

Ref. [14] produced the hitherto-best solution to the present problem. We, therefore, first compare the performance of the new algorithm against that of Ref. [14]. Subsequently, performance comparisons are presented with simulated annealing (SA) [10], genetic algorithm (GA) [12] and (1+1)-EA [12].

We consider two variants of the new algorithm: static penalty and adaptive penalty. For the new algorithm, the value of α is chosen as 0.999, and that of *evalFrac* is 0.05. The new algorithm and its three competitors being stochastic heuristics, multiple independent runs are needed for performance analysis. Four test-suites of each of the four competing algorithms are executed, where the (independent) runs in a test-suite are obtained by supplying different seeds to the random number generator. For simulated annealing and genetic algorithm, a single test-suite comprises five groups of 100 runs each, that is, a total of 500 runs, whereas for the new heuristic and (1+1)-EA, a suite consists of 100 runs.

Each run is allowed to proceed until *maxEvals* evaluations are consumed, and statistics are recorded for:

- best-of-run cost after *maxEvals* evaluations;
- the number of cost function evaluations required to reach a specified target cost for the first time in the run (this target is chosen to be slightly less than the cost of 13.7134 produced by the best solution in Ref. [14]); if the run fails to produce this cost by *maxEvals* evaluations, the run is marked as a “failure,” [4], otherwise it is a “success”.

The comparative performance is analyzed using the following metrics [4] on each test-suite:

- Solution quality, as given by the (i) best, (ii) worst, (iii) mean, and (iv) standard deviation (over 100 runs) of best-of-run costs obtained at a fixed number of function evaluations;
- The least amount of time to find an acceptable solution, as measured by the mean and standard deviation (over 100 runs; this number may be less than 100 if not all runs reach the stipulated target) of the minimum number of cost function evaluations required to reach, for the first time, a given (target) cost;
- Frequency of finding an acceptable solution, given by the number of “successful” runs out of the hundred.

An explanation of why only one metric between the first two in the above list does not suffice is in order. The solution quality corresponding to a given amount of computational effort is different from the computational effort needed to find a solution of a given quality in the context of comparative executions of stochastic heuristics (such as the present ones) where convergence to the global optimal solution is not guaranteed and where we often have to be happy with near-optimal or sub-optimal solutions. The word “solution” in this context encompasses true (i.e., globally optimum) solution(s), near-optimal (locally optimal) solution(s), and solutions (trial solutions) that are not even locally optimal. The first metric (in the itemized list above) fixes, in advance, an amount of effort (typically, a computational cost as measured in terms of the number of objective function evaluations needed) as an upper limit and then records the quality of the best solution that is obtained at the expense of those many function evaluations. The second metric, on the other hand, fixes, in advance, a quality (numerically measured in terms of the objective value of the proposed solution) as a lower limit and then records at what minimum effort that quality is achieved. The difference between the two metrics is important in comparative performance analysis of computational algorithms, because one algorithm may produce a high-quality solution fairly quickly and then stagnate for the rest of the run while its competitor may be sluggish for most of the run, only to produce a brilliant solution at the very end. Unless we use both the metrics, a complete picture of an algorithm’s performance cannot be obtained.

For each of the four test-suites of the static penalty version of the new algorithm, a single run (out of the hundred) is randomly chosen and the five best solutions (stack designs) produced by that single run are presented in Table 4 where the best solution published in Ref. [14] is also shown. (The best solutions in this table are not necessarily the best in the entire 100-run suite for a given suite; they are the best solutions in just one run in that suite.) The N_s, N_p, and A_{cell} values of the solution of Ref. [14] are taken from that source; the maximum power, the load voltage at maximum power, and the cost are computed as in Section 3.5 of the present paper. The new algorithm outperforms Ref. [14] in both the solution quality (the cost column in Table 4) and the number of function evaluations needed to produce the solution (the last column).

Table 4
Five best solutions from a single run of the new algorithm (static penalty) in each test-suite.

Run	*maxEvals*	*poolSize*	Solution			Cost	$P_{\text{load,max}}$, W	$V_{\text{load,maxpp}}$, V	Evaluations
			N_s	N_p	A_{cell}, cm^2				
1	1000	100	22	1	153.243	13.4455	206.4683	12.2292	648
			22	1	153.296	13.4887	206.5407	12.2335	659
			22	1	154.67	13.497	208.3920	12.2343	757
			22	1	156.038	13.5014	210.2352	12.2345	287
			22	1	154.227	13.506	207.7952	12.2352	873
2	2000	100	22	1	153.243	13.4455	206.4683	12.2292	648
			22	1	159.163	13.4585	214.4449	12.2299	1674
			22	1	156.885	13.4531	211.3756	12.2296	1310
			22	1	158.735	13.4791	213.8686	12.2320	1652
			22	1	156.02	13.4867	210.2107	12.2331	1088
3	3000	100	22	1	149.597	13.4353	201.5564	12.2286	2264
			22	1	150.513	13.4415	202.7903	12.2291	2133
			22	1	155.061	13.4468	208.9180	12.2292	1226
			22	1	152.342	13.4513	205.2547	12.2299	380
			22	1	158.261	13.463	213.2297	12.2305	685
4	4000	100	22	1	160.525	13.4583	216.2801	12.2298	1765
			22	1	155.986	13.4603	210.1645	12.2304	415
			22	1	158.713	13.462	213.8386	12.2303	1305
			22	1	160.985	13.4627	216.8997	12.2302	2106
			22	1	161.446	13.468	217.5209	12.2307	2831
Ref. [14]			21	1	156.25	13.7134	200.9524	11.6943	4000

To test whether the new method (with static penalty) is statistically significantly better than that in Ref. [14], we perform one-sample *t*-tests on the data in Table 4 under the assumption that the population distribution of the new method's solutions is normal; results of these tests are presented in Table 5. We test the hypothesis that the population corresponding to the new method's PEMFC design has a mean cost worse (i.e., greater) than or equal to the best cost reported in Ref. [14]. Thus, we test the null hypothesis

$$H_0 : \mu \geq \mu_0$$

against the one-sided alternative

$$H_1 : \mu < \mu_0,$$

where μ is the mean cost of the PEMFC design produced by the proposed method, and

$$\mu_0 = 13.7134.$$

We want to draw our conclusion such that the probability of a type-I error does not exceed 0.10. We choose a level of significance

$$\alpha \leq 0.10.$$

The population standard deviation can be estimated by the sample standard deviation, *S*. Since the probability of type-I error is greatest when $\mu = \mu_0$, we proceed as if we were testing the null hypothesis $\mu = \mu_0$ against the alternative hypothesis $\mu < \mu_0$ at the 0.10 level of significance [8]. The null hypothesis, then, must be rejected if

Table 5
Results of *t*-tests: new algorithm (static penalty) vs. Ref. [14]. Sample size = 5 and degrees of freedom = 4 in all cases.

Run	Mean	Std. dev.	*t*-statistic	90% confid. intvl.
1	13.4877	0.0244	−20.6836	(13.4644–13.5110)
2	13.4646	0.0176	−31.6099	(13.4478–13.4814)
3	13.4476	0.0105	−56.6045	(13.4376–13.4576)
4	13.4623	0.0036	−155.9657	(13.4589–13.4657)

Table 6
New heuristic (static penalty) (each row corresponds to 100 indepenedent runs).

maxEvals	*poolSize*	Best-of-run cost (summary of 100 runs)				Successes (target: cost<13.5)	
		Best	Worst	Mean	Std. dev.	Count	Mean cost
1000	100	13.441	34.67	16.245	5.21	5	13.4620
2000	100	13.4408	34.5776	16.1277	5.0850	7	13.4663
3000	100	13.435	34.57	16.11	5.12	8	13.4824
4000	100	13.4362	34.5263	16.0895	4.9697	7	13.4853

Table 7
Simulated annealing with static penalty (each row corresponds to 100 indepenedent runs).

maxEvals	*poolSize*	Init. temp.	Best-of-run cost (summary of 100 runs)				Successes (target: cost < 13.5)	
			Best	Worst	Mean	Std. dev.	Count	Mean cost
		10	13.455	42.52	16.85	6.40	4	
		50	13.465	35.53	16.48	5.65	3	
1000	100	100	13.459	38.66	16.50	5.79	4	13.4849
		1000	13.474	42.86	16.55	6.16	3	
		1.0e6	13.530	48.73	18.37	7.87	0	
		10	13.455	42.52	16.85	6.40	4	
		50	13.454	35.53	16.47	5.65	4	
2000	100	100	13.459	38.66	16.50	5.79	4	13.4829
		1000	13.468	42.86	16.55	6.16	5	
		1.0e6	13.473	48.73	16.31	5.99	5	
		10	13.455	42.52	16.85	6.40	4	
		50	13.454	35.53	16.47	5.65	4	
3000	100	100	13.459	38.66	16.50	5.79	4	13.4829
		1000	13.468	42.86	16.55	6.16	5	
		1.0e6	13.473	48.73	16.31	5.99	5	
		10	13.455	42.52	16.85	6.40	4	
		50	13.454	35.53	16.47	5.65	4	
4000	100	100	13.459	38.66	16.50	5.79	4	13.4829
		1000	13.468	42.86	16.55	6.16	5	
		1.0e6	13.473	48.73	16.31	5.99	5	

Table 8
Results of t-tests: new algorithm (with static penalty) vs. simulated annealing (with static penalty). Sample size = 5 and degrees of freedom = 4 in all cases.

maxEvals	*poolSize*	Mean	Std. dev.	t-statistic	90% confid. intvl.
1000	100	13.4770	0.0307	2.6224	(13.4477–13.5063)
2000	100	13.4620	0.0084	5.5902	(13.4540–13.4700)
3000	100	13.4610	0.0089	6.5323	(13.4525–13.4695)
4000	100	13.4600	0.0100	5.3218	(13.4505–13.4695)

Table 9
New heuristic with adaptive penalty (each row corresponds to 100 indepenedent runs).

maxEvals	*poolSize*	Best-of-run cost (summary of 100 runs)				Successes (target: cost<13.5)	
		Best	Worst	Mean	Std. dev.	Count	Mean cost
1000	100	13.4480	77.0357	16.7265	8.6198	4	13.4839
2000	100	13.4365	34.7069	16.0864	5.0963	6	13.4821
3000	100	13.4347	34.5328	15.8602	4.8677	10	13.4796
4000	100	13.4346	34.5453	16.0066	5.0298	9	13.4797

$$t < c,$$

where the test statistic t is given by

$$t = \frac{\overline{X} - \mu_0}{S/\sqrt{(n)}},$$

where $\overline{X}$ is the sample mean, n = sample size = 5, and c is a critical value obtained from the relation

$$P(t < c) = \alpha = 0.10.$$

Now, from the standard t-distribution table, we have

$$c = -1.533$$

for 5–1 = 4 degrees of freedom. From Table 5 we see that the t-statistic is less than c in all the cases. Therefore, we must reject the null hypothesis at the 0.10 level of significance. Table 5 also shows the 90% confidence interval of the mean best cost produced by the new method. All of the confidence intervals lie below the Mohamed-Jenkins best cost.

Table 6 shows the results of execution of four suites of the new algorithm (static penalty version), each suite comprising 100 independent runs. The best cost, worst cost, average cost, and the standard deviation of the 100 best-of-run solutions are presented. Both the solution quality and the consistency with which they are found improve with an increase in the number of evaluations allowed, a behavior that is expected of a randomized search heuristic. All entries in the "Best" column are numerically less (that is, better) than the best solution in Ref. [14].

Table 6 also presents the "success" rate or how frequently a desired cost is obtained. The target cost in each run in each 100-run suite is reached whenever the cost falls below (i.e., is better than) a fixed value for the first time in a run. (The target value is chosen as a cost lower (better) than the best cost reported in Ref. [14]). Because the cost is a floating-point number, not an integer, we cannot use an equality in the definition of the target. In the first suite, for example, the value of 5 in the "Count" column indicates that five runs out of a hundred produced a cost less than 13.5 by the end of the run (the remaining 95 runs never produced a cost of 13.5 or better in 1000 evaluations); the 5 runs that did succeed took an average of 629.2 evaluations (see Table 18 later in this section) to reach the target *for the first time*. The last column, "Mean cost", presents the average of the costs of the *first* target-meeting solutions from the successful runs (only the very first target-meeting solution being considered from each successful

Table 11
Results of t-tests: new algorithm (with adaptive penalty) vs. simulated annealing (with adaptive penalty). Sample size = 5 and degrees of freedom = 4 in all cases.

maxEvals	*poolSize*	Mean	Std. dev.	t-statistic	90% confid. intvl.
1000	100	13.4767	0.0082	7.8216	(13.4688–13.4845)
2000	100	13.4526	0.0057	6.3159	(13.4472–13.4580)
3000	100	13.4526	0.0057	7.0220	(13.4472–13.4580)
4000	100	13.4526	0.0057	7.0613	(13.4472–13.4580)

Table 10
Simulated annealing with adaptive penalty (each row corresponds to 100 indepenedent runs).

maxEvals	*poolSize*	Init. temp.	Best-of-run cost (summary of 100 runs)				Successes (target: cost < 13.5)	
			Best	Worst	Mean	Std. dev.	Count	Mean cost
		10	13.4690	76.8695	16.9680	8.6951	4	
		50	13.4731	76.4888	16.8196	8.4914	4	
1000	100	100	13.4743	76.7499	17.0701	9.0010	4	13.4877
		1000	13.4764	63.3905	16.5666	7.5408	3	
		1.0e6	13.4905	79.5083	18.6464	10.0095	1	
		10	13.4554	42.5244	16.7999	6.3589	4	
		50	13.4482	35.6786	16.3700	5.4310	4	
2000	100	100	13.4611	34.7313	16.4178	5.5418	5	13.4800
		1000	13.4513	35.4435	16.2072	5.1984	6	
		1.0e6	13.4472	42.4227	16.3544	5.6661	5	
		10	13.4554	42.5244	16.7999	6.3589	4	
		50	13.4482	35.6786	16.3700	5.4310	4	
3000	100	100	13.4611	34.7313	16.4178	5.5418	5	13.4800
		1000	13.4513	35.4435	16.2072	5.1984	6	
		1.0e6	13.4472	42.4227	16.3544	5.6661	5	
		10	13.4554	42.5244	16.7999	6.3589	4	
		50	13.4482	35.6786	16.3700	5.4310	4	
4000	100	100	13.4611	34.7313	16.4178	5.5418	5	13.4800
		1000	13.4513	35.4435	16.2072	5.1984	6	
		1.0e6	13.4472	42.4227	16.3544	5.6661	5	

Table 12
Genetic algorithm with static penalty (each row corresponds to 100 indepenedent runs).

maxEvals	*PopSize*	Generations	Crossover	Mutation	Best-of-run cost (summary of 100 runs)			Successes (target: < 13.5)	
					Best	Mean	Std. dev.	Count	Mean cost
1000	30	20	0.6	0.05	13.4779	19.1441	10.8827	2	13.4654
			0.7	0.005	13.4499	38.4856	22.4062	1	
			0.7	0.01	13.4499	31.3056	21.7365	2	
			0.8	0.01	13.6036	31.8345	20.1252	0	
			0.9	0.005	13.4567	32.3906	19.7207	1	
2000	40	30	0.6	0.05	13.4491	15.0526	6.6952	7	13.4766
			0.7	0.005	13.4898	26.7768	15.6620	1	
			0.7	0.01	13.4670	21.0918	12.5377	3	
			0.8	0.01	13.4689	19.6511	11.1000	1	
			0.9	0.005	13.4761	24.5101	11.7219	1	
3000	40	40	0.6	0.05	13.4491	14.5412	3.5617	7	13.4738
			0.7	0.005	13.4898	24.5617	13.9246	1	
			0.7	0.01	13.4581	19.6072	10.5742	4	
			0.8	0.01	13.4613	18.7009	9.6915	2	
			0.9	0.005	13.4567	22.5749	11.0708	2	
4000	50	45	0.6	0.05	13.4442	14.1297	2.2914	15	13.4834
			0.7	0.005	13.4604	18.8314	8.4060	4	
			0.7	0.01	13.4673	17.0143	6.0552	3	
			0.8	0.01	13.4782	16.1987	4.9404	7	
			0.9	0.005	13.4723	18.7952	7.1753	3	

Table 13
Results of *t*-tests: new algorithm (with static penalty) vs. genetic algorithm (with static penalty). Sample size = 5 and degrees of freedom = 4 in all cases.

maxEvals	*poolSize*	Mean	Std. dev.	*t*-statistic	90% confid. intvl.
1000	100	13.4876	0.0659	1.5812	(13.4248–13.5504)
2000	100	13.4702	0.0148	4.4419	(13.4561–13.4843)
3000	100	13.4630	0.0156	4.0135	(13.4481–13.4779)
4000	100	13.4645	0.0131	4.8306	(13.4520–13.4770)

run), the number of the successful runs being given in the “Count” column. For example, the mean cost of 13.4620 in the first row is obtained as the mean of the five solutions from five distinct (successful) runs out of the 100 runs in the first suite, where each of these five solutions is the first-ever solution to meet the target in the run, regardless of whether or not that run subsequently produced a better-quality solution.

Next, we study the relative performance of simulated annealing on this problem. Simulated annealing has no difference estimation step; therefore, for fair comparison, it is started, in each case, with the same point with which the new algorithm starts. Simulated annealing uses a temperature parameter that is initialized to a relatively high value and gradually decreased via the geometric schedule:

$$temperature = 0.99 * temperature.$$

The acceptance probability of a new point is given by

$$prob = e^{-d/temperature}$$

Table 14
Genetic algorithm with adaptive penalty (each row corresponds to 100 indepenedent runs).

maxEvals	*PopSize*	Generations	Crossover	Mutation	Best-of-run cost (summary of 100 runs)			Successes (target: < 13.5)	
					Best	Mean	Std. dev.	Count	Mean cost
1000	30	20	0.6	0.05	13.4432	19.9000	10.2593	5	13.4650
			0.7	0.005	13.4499	41.0601	22.8921	1	
			0.7	0.01	13.4499	33.0051	21.2640	1	
			0.8	0.01	13.4723	34.1800	20.6823	1	
			0.9	0.005	13.4567	37.5282	21.3904	1	
2000	40	30	0.6	0.05	13.4412	16.1963	6.1908	6	13.4691
			0.7	0.005	13.5481	28.5163	17.0902	0	
			0.7	0.01	13.4712	23.2383	13.7215	3	
			0.8	0.01	13.4432	22.7769	12.4293	2	
			0.9	0.005	13.5351	26.3677	12.8909	0	
3000	40	40	0.6	0.05	13.4412	15.8257	4.8658	7	13.4732
			0.7	0.005	13.4941	25.0664	14.2906	1	
			0.7	0.01	13.4669	21.1757	11.5537	5	
			0.8	0.01	13.4432	20.7075	9.9013	3	
			0.9	0.005	13.5075	24.3502	11.9836	0	
4000	50	45	0.6	0.05	13.4424	13.8508	1.3174	17	13.4805
			0.7	0.005	13.4489	20.9728	10.0436	4	
			0.7	0.01	13.4556	17.4829	6.2357	3	
			0.8	0.01	13.4408	16.9992	5.8925	8	
			0.9	0.005	13.4616	21.1321	8.9672	2	

Table 15
Results of t-tests: new algorithm (with adaptive penalty) vs. genetic algorithm (with adaptive penalty). Sample size=5 and degrees of freedom=4 in all cases.

maxEvals	*poolSize*	Mean	Std. dev.	t-statistic	90% confid. intvl.
1000	100	13.4538	0.0103	1.2591	(13.4440–13.4636)
2000	100	13.4870	0.0515	2.1929	(13.4379–13.5361)
3000	100	13.4699	0.0305	2.5821	(13.4408–13.4990)
4000	100	13.4497	0.0092	3.6674	(13.4409–13.4585)

Table 16
(1+1)-EA with static penalty (each row corresponds to 100 indepenedent runs).

maxEvals	*poolSize*	Best-of-run cost (summary of 100 runs)				# Successes (target: cost < 13.5)
		Best	Worst	Mean	Std. dev.	
1000	100	13.5182	62.5788	17.8733	8.3328	0
2000	100	13.5182	62.5788	17.8733	8.3328	0
3000	100	13.5182	62.4914	17.8711	8.3287	0
4000	100	13.5182	62.4914	17.8711	8.3287	0

Table 17
(1+1)-EA with adaptive penalty (each row corresponds to 100 indepenedent runs).

maxEvals	*poolSize*	Best-of-run cost (summary of 100 runs)				Successes (target: cost < 13.5)	
		Best	Worst	Mean	Std. dev.	Count	Mean cost
1000	100	13.4772	76.6891	17.2897	9.02510	2	13.4779
2000	100	13.4389	42.8807	17.0800	6.78241	1	13.4389
3000	100	13.4389	42.8807	17.0788	6.78304	1	13.4389
4000	100	13.4389	42.8807	17.0788	6.78304	1	13.4389

where d represents the difference between the costs of the "current" point and the "next," the "next" point being obtained by mutating the "current." For a fair comparison, all four algorithms in this study are implemented with the same mutation procedure.

The performance of simulated annealing (with static penalty) is shown in Table 7 where 100 independent runs are taken for each initial temperature in each suite (a suite has five different initial temperatures). The values in the last column are obtained by averaging over the successful runs from all 500 runs in each suite. All the four metrics—best, worst, mean, and standard deviation—are better (lower) for all cases in Table 6 than in Table 7. The success counts are also higher for the new method in all cases. With the exception of the *maxEvals* = 4000 case, the last column values are better for the new method than for the SA.

Table 8 presents t-test results of comparing SA's solution quality against that of the static penalty version of the new method. For each suite, the best of best-of-run costs of the new algorithm is compared against the mean of the five values representing the best of best-of-run costs of the SA. The new method outperforms SA on all the suites; the computed t is greater than $t_{0.10,\ 4\ \text{d.f.}}$=1.533 and the new method's solution lies below the 90% confidence interval.

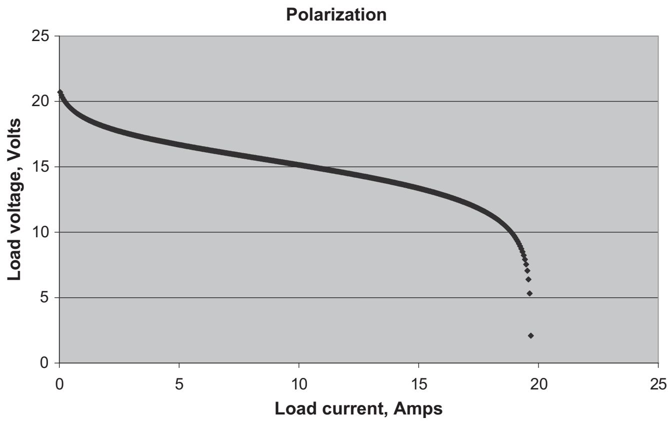

Fig. 1. PEMFC stack polarization.

Results for the adaptive penalty variant of the new heuristic are summarized in Table 9. Table 10 presents the corresponding results for simulated annealing with adaptive penalty. As in the static penalty case, here, too, the new method outperforms SA on both solution quality and success rate (the mean and standard deviation values are better for the SA in only one case). Table 11 shows that the differences in solution quality between the two algorithms are significant at the 0.01 level of significance.

Tables 12 and 14 show results produced by the two variants (static and adaptive penalties) of a standard genetic algorithm. The same penalty methods as those used in the new algorithm are used for cost computation in the GA. The different test suites correspond to different parameter settings (population size, maximum number of generations, probability of crossover, and probability of mutation) of the GA. A generational GA is used with binary tournament selection and 0.5-uniform crossover with one offspring being produced from two parents. Each of the three variables is mutated independently with the specified mutation probability.

The effective *maxEvals* values for the genetic algorithm are close, but not exactly identical, to the values shown in Tables 12 and 14 (these are the fixed values used for all the simulations in this paper). This is because we create a new point in the GA by a (probabilistic) crossover operation followed by a (probabilistic) mutation operation, each of which, if triggered, requires creating a new point. In the implementation of the GA used in this paper, mutation always consumes one evaluation (a new copy of the input point is produced as the output of the mutation step, even if no variable is changed by mutation). In the implementation of the new heuristic and of the SA (and of the (1+1)-EA, discussed later in this section), no attempt is made to avoid re-evaluation of copies of previously generated points.

Only in three cases out of the twenty is the mean best-of-run cost better for the GA (static penalty) (Table 12) than for the new method (static penalty) (Table 6). The best cost in a 100-run suite, however, is in all cases better for the new method (static) than for

Table 18
Comparative study of earliest evaluations to target (cost<13.5) (SD = standard deviation).

maxEvals	New (static)		New (adaptive)		GA (static)		GA (adaptive)		SA (static)		SA (adaptive)	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
1000	629.2	168.9	662.0	97.9	424.2	259.3	434.8	172.3	369.9	195.7	394.3	204.9
2000	943.3	519.8	654.3	267.2	1333.2	419.5	1448.0	359.4	726.6	528.8	608.3	367.2
3000	574.3	405.9	1042.0	381.4	1568.4	643.8	1747.9	550.7	726.6	528.8	608.3	367.2
4000	1293.9	1026.2	1152.4	584.4	2105.0	1008.7	1823.2	781.8	726.6	528.8	608.3	367.2

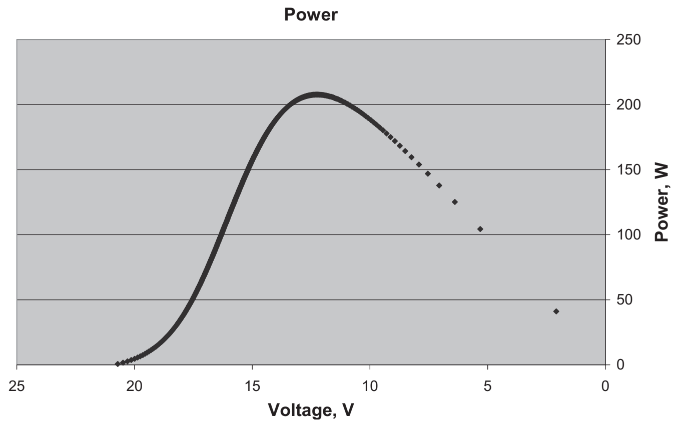

Fig. 2. PEMFC stack voltage–power plot.

the GA (static). Table 13 shows that the *t*-statistic is always greater than $t_{0.10}$ at four degrees of freedom.

A comparison of Tables 9 and 14 shows that the "Best" value is superior for the new algorithm (adaptive penalty) than for the GA (adaptive penalty) for all cases except one (the first row in the GA table). On the "Mean" and the "Std. dev." columns, the new method is better on all cases except two. The *t*-test results in Table 15 show only one case where the computed *t* is less than $t_{0.10}$.

Simulation results of the static and adaptive penalty variants of the (1+1)-EA are presented in Tables 16 and 17. A comparison of these results with the data in Tables 6 and 9 show that the new algorithm outperforms the (1+1)-EA on all counts with two exceptions: (i) the "Worst" value in the first suite in Table 17 is better than the corresponding value in Table 9, (ii) the "Mean cost" values in Table 17 are better than their counterparts in Table 9. Given the definition of the "Worst" column and the smallness of the "Count" values of (1+1)-EA (adaptive), neither of these two exceptions seems to be of much importance in the comparative study.

Table 18 presents a study of how fast the different algorithms produce an acceptable solution. Recall that the number of runs used to calculate the means and standard deviations in this table are indicated in the "Count" column of Tables 6, 7, 9, 10, 12 and 14. For example, the value 629.2 in the first row of Table 18 is obtained as the average of the five (see the "Count" column of Table 6) successful runs (the five target-meeting solutions that produced 629.2 as the mean number of evaluations to reach the target also had a mean cost of 13.4620, recorded in the last column of Table 6), while genetic algorithm's 1823.2 in the last row of Table 18 is calculated as the average of 34 target-meeting solutions (with a mean cost of 13.4805) from the last suite in Table 14.

The data in Table 18 record (the mean and standard deviation of) the number of evaluations corresponding to only the very first (earliest) appearance in a run of a solution that reaches the specified target (cost < 13.5) in that run; that is, the data do not take into account other, possibly better, target-reaching solutions that may have been produced by the algorithm later in the same run. Thus the count of 5 in the first row of the "Count" column of Table 6 shows just the number of independent runs (in a 100-run suite) that found at least one less-than-13.5 cost by 1000 evaluations; some or all of those five runs probably generated more than one solution with a less-than-13.5 cost by 1000 evaluations, but only the first (fewest-evaluation-consuming) solution from each run was considered in calculating the mean and the standard deviation of the number of evaluations needed to reach the target.

Table 18 shows the summary of 100 independent runs for either variant of the new heuristic and of 500 independent runs for each of the other two methods (genetic algorithm and simulated annealing). The (1+1)-EA (adaptive penalty) produced only one successful run out of the hundred in three suites and two in the fourth; its static penalty variant did not produce any successful run. The (1+1)-EA is therefore not included in this table.

On the metric on earliest hitting of the target, in the majority of cases, the new heuristic outperforms genetic algorithm but is outperformed by SA. However, in a pairwise comparison between the new method and the SA (Table 6 vs. 7; Table 9 vs. 10), the solutions used in the earliest-hitting calculations have a mean cost that is better for the new method six times out of the eight. In a similar cost comparison between the new method and the GA (Table 6 vs. 12; Table 9 vs. 14), the new method outperforms the GA three times out of the eight. Thus it can be argued that the smaller hitting time is achieved at the cost of reduced solution quality. Again, given the SA's apparent failure to capitalize on the extra evaluations available (see the last three rows for SA in Table 18), the relatively quick hitting of the target by SA seems to be of limited value.

The solutions presented in the tables in this paper are all feasible.

Figs. 1 and 2 show polarization and voltage-vs-power characteristics of a PEMFC stack design obtained with a typical run ($N_s = 22$, $N_p = 1$, A_{cell}=154.164 cm^2, *maxEvals*=500, *poolSize*=100, cost = 13.456, $P_{load,max}$ = 207.71 Watts, $V_{load,maxpp}$ = 12.23 V) of the proposed method (with static penalty).

6. Conclusion

This paper developed a new probabilistic heuristic for PEMFC stack design. The problem addressed analytically intractable parameter optimization involving a mix of discrete and continuous (real) variables, and is thus representative of several types of open problems in modeling and design of fuel cell-based systems. The use of a novel transition probability function led to an efficient search for an optimal configuration. Two variants of the new method were used: static penalty and adaptive penalty. Simulation results produced by the algorithm were seen to be superior to the best solution published in the literature so far. The proposed approach statistically significantly outperformed three of its competitors, namely, genetic algorithm, simulated annealing, and (1+1)-EA, in the majority of test cases. The relative performance of the competing algorithms was studied using three metrics:

- The solution quality obtained at a pre-determined computational cost;
- The computational cost needed to find a solution of a pre-determined quality;
- The frequency with which the algorithm finds an acceptable solution.

The adaptive penalty version of the new algorithm was seen to perform better than its static-penalty counterpart. A Markov chain analysis of the new algorithm led to a theoretical upper bound on the expected time to find the global optimum solution.

Acknowledgments

Thanks to the anonymous reviewers for their comments. The author is grateful to Ms. S. Cedergreen of Parkway Central High School, Professor C. L. Sabharwal of Missouri University of Science & Technology, Professor S. Iyer of Washington University in St. Louis, Professor J. Yukna of Maryville University, Professor N. Marić of University of Missouri - St. Louis, and Professor C. Ash of University of Illinois at Urbana-Champaign for their help and guidance.

References

[1] H.J.C. Barbosa, A.C.C. Lemonge, An Adaptive Penalty Method for Genetic Algorithms in Constrained Optimization Problems, in: Hitoshi Iba (Ed.), Frontiers in Evolutionary Robotics, I-Tech Education and Publishing, Vienna, 2008, pp. 9–34.

[2] J.C. Bean, A.B. Hadj-Alouane, A Dual Genetic Algorithm for Bounded Integer Programs. TR 92-53, University of Michigan, 1992.

[3] U.K. Chakraborty, Energy 34 (2009) 740–751.

[4] U.K. Chakraborty, T.E. Abbott, S. Das, Energy 40 (2012) 387–399.

[5] F. Friede, S. Raël, B. Davat, IEEE Transactions on Power Electronics 19 (2004) 1234–1241.

[6] Fuel Cell Handbook, fifth ed., EG&G Services Parsons Inc, US Department of Energy, 2000.

[7] M. Gen, R. Cheng, A Survey of Penalty Techniques in Genetic Algorithms, Proceedings of IEEE International Conference on Evolutionary Computation, 1996, 804–809.

[8] R.A. Johnson, Miller & Freund's Probability and Statistics for Engineers, sixth ed., Pearson, 2000.

[9] J.G. Kemeny, J.L. Snell, Finite Markov Chains, Van Nostrand, Princeton, 1960.

[10] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Science 220 (1983) 671–680.

[11] J. Larminie, A. Dicks, Fuel Cell Systems Explained, second ed., Wiley, Chichester, 2003.

[12] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, third ed., Springer, Berlin, Heidelberg, 1996.

[13] Z. Michalewicz, D.B. Fogel, How to Solve It: Modern Heuristics, Springer, Berlin, Heidelberg, 2000.

[14] I. Mohamed, N. Jenkins, Journal of Power Sources 131 (2004) 142–146.

[15] B. Tessema, G.G. Yen, IEEE Transactions on Systems, Man, and Cybernetics – Part A 39 (2009) 565–578.